

**This Page is Inserted by IFW Indexing and Scanning  
Operations and is not part of the Official Record**

**BEST AVAILABLE IMAGES**

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ **BLACK BORDERS**
- ☐ **IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**
- ☐ **FADED TEXT OR DRAWING**
- ☐ **BLURRED OR ILLEGIBLE TEXT OR DRAWING**
- ☐ **SKEWED/SLANTED IMAGES**
- ☐ **COLOR OR BLACK AND WHITE PHOTOGRAPHS**
- ☐ **GRAY SCALE DOCUMENTS**
- ☐ **LINES OR MARKS ON ORIGINAL DOCUMENT**
- ☐ **REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**
- ☐ **OTHER:** \_\_\_\_\_

**IMAGES ARE BEST AVAILABLE COPY.**

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.



# UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE  
United States Patent and Trademark Office  
Address: COMMISSIONER FOR PATENTS  
P.O. Box 1450  
Alexandria, Virginia 22313-1450  
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/542,189	04/04/2000	Gordon Taylor Davis	RAL9-2000-0008-US1	5890
25299	7590	09/16/2004	EXAMINER	
IBM CORPORATION PO BOX 12195 DEPT 9CCA, BLDG 002 RESEARCH TRIANGLE PARK, NC 27709			HUISMAN, DAVID J	
			ART UNIT	PAPER NUMBER
			2183	

DATE MAILED: 09/16/2004

Please find below and/or attached an Office communication concerning this application or proceeding.

# Office Action Summary

Application No.

09/542,189

Applicant(s)

DAVIS ET AL.

Examiner

David J. Huisman

Art Unit

2183



-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

## Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
  - If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
  - If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
  - Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133).
- Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

## Status

- 1) ☒ Responsive to communication(s) filed on 06 July 2004.
- 2a) ☐ This action is FINAL. 2b) ☒ This action is non-final.
- 3) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

## Disposition of Claims

- 4) ☒ Claim(s) 1-31 is/are pending in the application.
- 4a) Of the above claim(s) \_\_\_\_\_ is/are withdrawn from consideration.
- 5) ☐ Claim(s) \_\_\_\_\_ is/are allowed.
- 6) ☒ Claim(s) 1-31 is/are rejected.
- 7) ☐ Claim(s) \_\_\_\_\_ is/are objected to.
- 8) ☐ Claim(s) \_\_\_\_\_ are subject to restriction and/or election requirement.

## Application Papers

- 9) ☐ The specification is objected to by the Examiner.
- 10) ☒ The drawing(s) filed on 04 April 2000 is/are: a) ☒ accepted or b) ☐ objected to by the Examiner.
- Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).
- Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
- 11) ☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

## Priority under 35 U.S.C. § 119

- 12) ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
- a) ☐ All b) ☐ Some \* c) ☐ None of:
- ☐ Certified copies of the priority documents have been received.
  - ☐ Certified copies of the priority documents have been received in Application No. \_\_\_\_\_.
  - ☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

\* See the attached detailed Office action for a list of the certified copies not received.

## Attachment(s)

- |   |   |
|---|---|
| 1) <input checked="" type="checkbox"/> Notice of References Cited (PTO-892)             | 4) <input type="checkbox"/> Interview Summary (PTO-413)                     |
| 2) <input type="checkbox"/> Notice of Draftsperson's Patent Drawing Review (PTO-948)    | Paper No(s)/Mail Date. _____  |
| 3) <input type="checkbox"/> Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) | 5) <input type="checkbox"/> Notice of Informal Patent Application (PTO-152) |
| Paper No(s)/Mail Date _____   | 6) <input type="checkbox"/> Other: _____                                    |

### **DETAILED ACTION**

1. Claims 1-31 have been examined.

#### ***Papers Submitted***

2. It is hereby acknowledged that the following papers have been received and placed of record in the file: Extension of Time and Appeal Brief as received on 7/6/2004.

#### ***Response to Arguments***

3. In view of the appeal brief filed on July 6, 2004, PROSECUTION IS HEREBY REOPENED. A new ground(s) of rejection is set forth below.

To avoid abandonment of the application, appellant must exercise one of the following two options:

(1) file a reply under 37 CFR 1.111 (if this Office action is non-final) or a reply under 37 CFR 1.113 (if this Office action is final); or,

(2) request reinstatement of the appeal.

If reinstatement of the appeal is requested, such request must be accompanied by a supplemental appeal brief, but no new amendments, affidavits (37 CFR 1.130, 1.131 or 1.132) or other evidence are permitted. See 37 CFR 1.193(b)(2).

#### ***Claim Objections***

4. Claim 28 is objected to because of the following informalities: In line 9, replace "FIFO" with --the FIFO--. Appropriate correction is required.

***Claim Rejections - 35 USC § 112***

5. The following is a quotation of the second paragraph of 35 U.S.C. 112:

The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

6. Claim 1 recites the limitation "the queue" in lines 7 and 8. There is insufficient antecedent basis for this limitation in the claim because no previous mention of a queue is made. Instead, the step of queuing is mentioned but it should be realized that queuing can be associated with a single queue or multiple queues.
7. Claim 2 recites the limitation "the original thread" in line 3. There is insufficient antecedent basis for this limitation in the claim.
8. Claim 11 recites the limitation "the queue" in lines 7 and 8. There is insufficient antecedent basis for this limitation in the claim because no previous mention of a queue is made. Instead, the step of queuing is mentioned but it should be realized that queuing can be associated with a single queue or multiple queues.
9. Claim 12 recites the limitation "the original thread" in line 3. There is insufficient antecedent basis for this limitation in the claim.
10. Claim 14 recites the limitation "the control transfer means" in line 1. There is insufficient antecedent basis for this limitation in the claim.
11. Claim 26 recites the limitation "the buffer" in lines 7, 9, and 10. There is insufficient antecedent basis for this limitation in the claim because a prefetch buffer from line 4 may be read as being different from the buffer mentioned in line 5.

*Claim Rejections - 35 USC § 102*

12. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless –

(e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

13. Claims 11-16 and 19-23 are rejected under 35 U.S.C. 102(e) as being anticipated by Parady, U.S. Patent No. 5,933,627 (as applied in the previous Office Action).

14. Referring to claim 11, Parady has taught using multiple threads to access data, including:

a) a CPU configured with multiple instruction execution threads as independent processes in a sequential time frame. See Fig.3 and note the implementation of multiple threads. These threads are independent because while one thread is stalled, the system will begin execution of another thread (see the abstract). In addition, each of the threads has its own dedicated resources, such as the registers shown in Fig.3.

b) a thread execution control for:

b1) queuing the multiple execution threads to have overlapping access to the accessible data. See Fig.3 and note that each of the threads are queued in instruction buffers 102-108. Also, when threads are switched, they have overlapping access to data.

b2) executing a first thread in the queue. See Fig.3 and note that one of the four threads will initially be executed when its instructions are sent to the dispatch unit.

b3) transferring control of the execution to the next thread in the queue upon the occurrence of an event that causes execution of the first thread to stall. See the abstract.

Art Unit: 2183

15. Referring to claim 12, Parady has taught a system as described in claim 11. Parady has further taught that the thread execution control includes control logic for temporarily transferring the control to the next thread when execution stalls due to a short latency event, and for returning control to the original thread when the latency event is completed. See the abstract. It should be noted that “short” is a relative word, i.e., a “short” event may be a “long” event without a “long” event to compare with. Consequently, Parady reads on applicant’s claim. In addition, it should be noted that control will eventually be transferred back to the first thread. See column 5, lines 13-14. The claim does not explicitly state that control is returned to the original thread immediately upon completion of the event.

16. Referring to claim 13, Parady has taught a system as described in claim 12. Parady has further taught that a processor instruction is encoded to select a short latency event. Recall that Parady has taught thread-switching when a cache-miss occurs. See the abstract. It is inherent that such a short latency event (i.e. a cache miss) would result from trying to load data that is not in the cache, for example. It is the load instruction’s encoding that causes the processor to access the cache and select a short latency event upon a cache miss.

17. Referring to claim 14, Parady has taught a system as described in claim 11. Parady has further taught that the control transfer means includes the means for transferring full control of the execution to the next thread when execution of the first thread stalls due to a long latency event. See the abstract. It should be noted that “long” is a relative word, i.e., a “long” event may be a “short” event without a “short” event to compare with. Consequently, Parady reads on applicant’s claim.

Art Unit: 2183

18. Referring to claim 15, Parady has taught a system as described in claim 14. Parady has further taught that a processor instruction is encoded to select a long latency event. Recall that Parady has taught thread-switching when a cache-miss occurs. See the abstract. It is inherent that such a short latency event (i.e. a cache miss) would result from trying to load data that is not in the cache, for example. It is the load instruction's encoding that causes the processor to access the cache and select a short latency event upon a cache miss.

19. Referring to claim 16, Parady has taught a system as described in claim 11. Parady has further taught means to queue the threads to provide rapid distribution of access to shared memory. See Fig.1 and Fig.5 and note that at the very least, the threads share data cache 56/166.

20. Referring to claim 19, Parady has taught a system as described in claim 11. Parady has further taught that the processor uses zero overhead to switch execution from one thread to the next. Parady makes no mention of switch overhead. Consequently, Parady has taught that the switching requires no overhead.

21. Referring to claim 20, Parady has taught a system as described in claim 19. Parady has further taught that each thread is given access to an array of general purpose registers and local data storage to enable switching with zero overhead. See Fig.3 and note that each thread has access to its own set of integer and register files 48 and 50 and PA registers 110.

22. Referring to claim 21, Parady has taught a system as described in claim 20. Parady has further taught that the local data storage is made available to the processor by providing one address bit under the control of the thread execution control logic and by providing the remaining address bits under the control of the processor. See Fig.3 and Fig.4. It should be noted that in order to use the appropriate register file, a thread number must be provided via field



Art Unit: 2183

118 shown in Fig.4. These 2 bits (which include one address bit) will select which register file will be used, for instance. In addition, it is inherent that the processor will address the appropriate registers during execution, i.e., a register address must be specified.

23. Referring to claim 22, Parady has taught a system as described in claim 20. Parady has further taught that the processor is capable of simultaneously addressing multiple register arrays, and the thread execution control logic includes a selector to select which array will be delivered to the processor for a given thread. See Fig.7 and note that although multiple shadow files may be addressed simultaneously, only one will be enabled, causing that enabled data to be sent to the processor via selector 192.

24. Referring to claim 23, Parady has taught a system as described in claim 20. Parady has further taught that the local data storage is fully addressable by the processor and the thread execution control has no address control over the local data stored or the register arrays. See Fig.3, component 56, and note that the data cache inherently has locations which are addressable by the processor. Also, a plain data cache access has nothing to do with what thread is being executed. For instance, if data needs to be loaded from the cache, the load address is all that is needed to access the cache, not the thread number. Parady has further taught that an index register is contained within the register array. See Fig.3, component 110. Each of these registers is an index register in that they hold indexes into instruction memory from which thread instructions will be fetched.

***Claim Rejections - 35 USC § 103***

25. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

26. Claims 1-6 and 9-10 are rejected under 35 U.S.C. 103(a) as being unpatentable over Parady, as applied above, in view of Kruse and Ryba, "Data Structures And Program Design in C++," 1999 (herein referred to as Kruse).

27. Referring to claim 1, Parady has taught a use of multiple threads including the steps of:

a) providing multiple instruction execution threads as independent processes in a sequential time frame. See Fig.3 and note the implementation of multiple threads. These threads are independent because while one thread is stalled, the system will begin execution of another thread (see the abstract). In addition, each of the threads has its own dedicated resources, such as the registers shown in Fig.3.

b) queuing the multiple execution threads. See Fig.3 and note that each of the threads are queued in instruction buffers 102-108.

c) executing a first thread in the queue. See Fig.3 and note that one of the four threads will initially be executed when its instructions are sent to the dispatch unit.

d) transferring control of the execution to the next thread in the queue upon the occurrence of an event that causes execution of the first thread to stall. See the abstract.

e) Parady has not explicitly taught that the multiple threads have overlapping access to the accessible data available in said tree search structure. However, Kruse has taught that search

Art Unit: 2183

trees are quick and efficient for inserting, deleting, and searching for data. See section 10.2 on pages 444-445. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Parady such that the threads have access to data available in a tree search structure. Again, one would be motivated to have such a structure because they allow for fast searching, inserting, and deleting of data.

28. Referring to claim 2, Parady in view of Kruse has taught a use as described in claim 1. Parady has further taught that the control of the execution is temporarily transferred to the next thread when execution stalls due to a short latency event, and the control is returned to the original thread when the event is completed. See the abstract. It should be noted that “short” is a relative word, i.e., a “short” event may be a “long” event without a “long” event to compare with. Furthermore, applicant’s “short latency event” is not defined as being 25 cycles or less as applicant argues on page 18 of the appeal brief. Instead, page 4, lines 8-12, of applicant’s specification state “If the latency event is programmed to be short, e.g., 25 machine cycles or less...”. Using an example of 25 machine cycles does not exclude the short latency event from requiring some other amount of cycles larger than 25. Consequently, Parady reads on applicant’s claim. In addition, it should be noted that control will eventually be transferred back to the first thread. See column 5, lines 13-14. The claim does not explicitly state that control is returned to the original thread immediately upon completion of the event.

29. Referring to claim 3, Parady in view of Kruse has taught a use as described in claim 2. Parady has further taught that a processor instruction is encoded to select a short latency event. Recall that Parady has taught thread-switching when a cache-miss occurs. See the abstract. It is inherent that such a short latency event (i.e. a cache miss) would result from trying to load data

Art Unit: 2183

that is not in the cache, for example. It is the load instruction's encoding that causes the processor to access the cache and select a short latency event upon a cache miss.

30. Referring to claim 4, Parady in view of Kruse has taught a use as described in claim 1. Parady has further taught that full control of the execution is transferred to the next thread when execution of the first thread stalls due to a long latency event. See the abstract. It should be noted that "long" is a relative word, i.e., a "long" event may be a "short" event without a "short" event to compare with. Furthermore, applicant's "long latency event" is not defined as being greater than 25 cycles as applicant argues on page 19 of the appeal brief. Instead, page 4, lines 8-12, of applicant's specification state "On the other hand, if the latency event is programmed to be longer, e.g. over 25 cycles...". Using an example of being greater than 25 machine cycles does not exclude the long latency event from requiring some other amount of cycles less or equal to 25. Consequently, Parady reads on applicant's claim.

31. Referring to claim 5, Parady in view of Kruse has taught a use as described in claim 4. Parady has further taught that a processor instruction is encoded to select a long latency event. Recall that Parady has taught thread-switching when a cache-miss occurs. See the abstract. It is inherent that such a short latency event (i.e. a cache miss) would result from trying to load data that is not in the cache, for example. It is the load instruction's encoding that causes the processor to access the cache and select a short latency event upon a cache miss.

32. Referring to claim 6, Parady in view of Kruse has taught a use as described in claim 1. Parady has further taught queuing the threads to provide rapid distribution of access to shared memory. See Fig.1 and Fig.5 and note that at the very least, the threads share data cache 56/166.

Art Unit: 2183

33. Referring to claim 9, Parady in view of Kruse has a taught a use as described in claim 1. Parady has further taught that the threads are used with zero overhead to switch execution from one thread to the next. Parady makes no mention of switch overhead. Consequently, Parady has taught that the switching requires no overhead.

34. Referring to claim 10, Parady in view of Kruse has a taught a use as described in claim 9. Parady has further taught that each thread is given access to general purpose registers and local data storage to enable switching with zero overhead. See Fig.3 and note that each thread has access to its own set of integer and register files 48 and 50 and PA registers 110.

35. Claim 8 is rejected under 35 U.S.C. 103(a) as being unpatentable over Parady in view of Kruse, as applied above, and further in view of Flynn et al., U.S. Patent No. 6,052,708 (herein referred to as Flynn).

36. Referring to claim 8, Parady in view of Kruse has taught a use as described in claim 1. Parady has further taught a separate instruction buffer for each execution thread. See Fig.3. Parady in view of Kruse has not explicitly taught collecting instructions in a prefetch buffer for its execution thread when the thread is idle and when the instruction bandwidth is not being fully utilized. However, Flynn has taught such a concept. See column 4, lines 43-53. Note that when a thread is inactive, a thread's instructions may be fetched so that when the thread is made active, the instruction will be ready for dispatch immediately, thereby increasing efficiency. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Parady in view of Kruse to prefetch instructions for an inactive thread into its prefetch buffer.

37. Claim 18 is rejected under 35 U.S.C. 103(a) as being unpatentable over Parady, as applied above, in view of Flynn, as applied above.

38. Referring to claim 18, Parady has taught a system as described in claim 11. Parady has further taught a separate instruction buffer for each execution thread. See Fig.3. Parady in view has not explicitly taught means for collecting instructions in a prefetch buffer for its execution thread when the thread is idle and when the instruction bandwidth is not being fully utilized. However, Flynn has taught such a concept. See column 4, lines 43-53. Note that when a thread is inactive, a thread's instructions may be fetched so that when the thread is made active, the instruction will be ready for dispatch immediately, thereby increasing efficiency. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Parady to prefetch instructions for an inactive thread into its prefetch buffer.

39. Claim 26 is rejected under 35 U.S.C. 103(a) as being unpatentable over Parady in view of Lee et al., U.S. Patent No. 5,404,560 (as applied in the previous Office Action and herein referred to as Lee), and further in view of Flynn, as applied above.

40. Referring to claim 26:

a) Parady has taught associating each thread with a buffer. See Fig.3 and column 5, lines 6-10.

b) Parady has not explicitly taught determining whether a buffer associated with an execution thread is full. However, Lee has taught such a concept. See column 13, lines 52-58. A person of ordinary skill in the art would have recognized that the buffer should be checked to see if it is full. By doing this, the processor can determine when it should stop fetching instructions to fill

Art Unit: 2183

the buffer. If the processor did not check to see if the buffer was full, then instructions would continue to be fetched and since no room is available for storage in the prefetch buffer, another instruction would have to be overwritten. Therefore, in order to ensure that buffered instructions are not overwritten, it would have been obvious to one of ordinary skill in the art at the time of the invention to determine whether a buffer associated with an execution thread is full.

c) Parady in view of Lee has not explicitly taught that during periods that the buffer is not being used by an active execution thread, enabling the buffer to prefetch instructions for the execution thread. However, Flynn has taught such a concept. See column 4, lines 43-53. Note that when a thread is inactive, a thread's instructions may be fetched so that when the thread is made active, the instruction will be ready for dispatch immediately, thereby increasing efficiency. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Parady's buffers to actually be prefetch buffers in order to prefetch instructions for an inactive thread.

d) Parady in view of Lee and further in view of Flynn has further taught determining whether the thread associated with the buffer is active. More specifically, if the system is to fetch instructions for inactive threads, then the system must inherently be able to determine whether or not a thread is active.

41. Claims 27-29 and 31 are rejected under 35 U.S.C. 103(a) as being unpatentable over Joy et al., U.S. Patent No. 6,341,347 (as applied in the previous Office Action and herein referred to as Joy), in view of Kruse, as applied above, and further in view of Anderson et al., U.S. Patent No. 5,613,114 (herein referred to as Anderson). In addition, The Free On-Line Dictionary Of

Art Unit: 2183

Computing (FOLDOC), 1999, is cited as extrinsic evidence showing the meaning of “FIFO” (first-in first-out) and attached to this Office Action.

42. Referring to claim 27, Joy has taught a thread execution control useful for the efficient execution of independent threads comprising:

a) a plurality of thread control state machines, one for each thread. See Fig.3, components 310 and 312.

b) Joy has further taught that one of multiple threads may be selected for execution based on priority. See column 4, lines 20-22. Joy has not explicitly taught a priority FIFO buffer for storing thread numbers. However, Kruse has taught the well-known concept of a priority queue (also known as a priority FIFO since “FIFO” and “queue” are interchangeable according to FOLDOC). A priority FIFO, according to Kruse on pages 369-370, is specifically useful in time-sharing computer systems, which have a number of tasks (threads), each of the tasks having a corresponding priority. The priority FIFO allows for the finding and removal of the entry having the highest priority. Consequently, since Joy has taught one of multiple tasks are selected based on priority, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy to include a priority FIFO to hold these tasks because Kruse has taught that the priority FIFO is a structure which is able to provide such functionality. In addition, although Parady in view of Kruse has not taught actually storing thread numbers in the priority FIFO, Anderson has taught such a concept. See column 6, lines 42-45. A person of ordinary skill in the art would’ve recognized that there must be an efficient way to distinguish the tasks from each other. Anderson has taught that thread Ids (numbers) are stored in the priority FIFO. This is applicable to Joy because Joy employs thread Ids (TIDs). See Fig.5, lines 407, for



Art Unit: 2183

instance. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy in view of Kruse such that thread numbers are stored in the priority FIFO.

c) an arbiter for determining the thread execution priority among multiple threads based upon signals outputted from the FIFO buffer and the state machine. It is inherent that if one of a plurality of threads is selected, then some component, more specifically, an arbiter, must determine which thread to select. The highest-priority thread that is awaiting processing will be selected for execution. See column 4, lines 17-27. This selection is based on signals from the state machines and signals from the FIFO. More specifically, the state machines provide instructions (i.e., instruction signals) to the pipeline (Fig.3 and column 9, lines 22-25) which will cause the pipeline to stall or not stall (for instance, if the instruction is a load, it will cause the processor to either miss or hit the cache). In addition, the state machines define the state of the thread (column 8, lines 27-33). Therefore, signals will be outputted which signal that a thread is in the ready state. Once a switch is necessary based at least in part on the signals from the state machines, the priority value signals stored in the FIFO must be available so that the highest priority thread that is ready is selected.

43. Referring to claim 28, Joy in view of Kruse and further in view of Anderson has taught a thread execution control as described in claim 27.

a) Joy in view of Kruse in view of Anderson has further taught means for loading a thread number into the FIFO when a packet is dispatched to the processor. The nature of a FIFO (queue) is to enqueue data (in this case, the thread number), which results in loading a thread

Art Unit: 2183

number into the FIFO. This would be done when there is work to be performed by the loaded thread. See page 153 of Kruse.

b) Joy in view of Kruse in view of Anderson has further taught means for unloading a thread number from the FIFO when a packet has been enqueued for transmission. When the thread is selected to perform work it would be removed from the queue, as this is the nature of a FIFO. See page 153 and 369 of Kruse.

c) Joy in view of Kruse in view of Anderson has further taught thread number transfer from highest priority to lowest priority in the FIFO when a long latency event occurs. In the case where the thread priority decreases from front to rear of queue, when the highest priority thread is removed from the front, and it encounters a long latency event, then it will be reinserted at the end, where it must wait until it reaches the front of the queue to regain highest priority.

d) Joy in view of Kruse in view of Anderson has further taught thread outlets of the FIFO used to determine priority depending on the length of time a thread has been in the FIFO. Again, the inherent nature of a FIFO, according to page 153 of Kruse, is that entries are inserted at the rear and removed from the front. Therefore, the thread removed will have spent the longest amount of time in the queue because it has traveled all the way from the rear to the front (whereas the highest priority thread has only traveled from the rear to (front-1) position).

44. Referring to claim 29, Joy in view of Kruse in view of Anderson has taught a thread execution control as described in claim 27. Joy has further taught that the arbiter controls the priority of execution of multiple independent threads based on the Boolean expression recited in claim 29 comprising:

Art Unit: 2183

a) determining whether a request R is active or inactive. From Fig. 2B and column 6, lines 59-66, it can be seen that when all threads are stalled, there are no requests for execution and hence an idle processor (as denoted by reference number 240). If a thread's stall event has completed, then it is ready for execution, and it will request and make itself available for execution. It should be noted that an implicit request is made when a thread finishes its long latency event, signaling that it is ready to be executed again. In other words, a non-stalled thread is actively requesting execution while a stalling thread is inactively requesting execution.

b) determining the priority of the threads. See column 4, lines 20-25.

c) matching the request R with the corresponding thread P. Clearly, only requesting (ready) threads may be executed. Of the requesters, only the highest priority will be chosen. See column 4, lines 20-25.

d) granting a request for execution if the request is active and if the corresponding thread P has the highest priority. Again, a non-stalled thread is ready for execution and therefore actively requesting execution control. From the group of threads that are not stalled (requesting execution), the one with the highest priority will be chosen. See column 4, lines 20-25.

It should be noted that the formula of claim 29 represents the algorithm set forth in steps (a)-(d) of claim 29, and Joy has taught steps (a)-(d), then Joy has also taught the formula even though it is not explicitly stated.

45. Referring to claim 31, Joy in view of Kruse in view of Anderson has taught a thread executing control according to claim 28. Joy has further taught means to detect occurrence of latency events. See column 7, lines 1-4. Joy in view of Kruse in view of Anderson has not explicitly taught that this means is included within the FIFO. However, as shown in In re

Art Unit: 2183

Larson, 144 USPQ 347 (CCPA 1965), to make integral is generally not given patentable weight or would have been an obvious improvement. That is, even if this means is outside of the FIFO in Joy, it still anticipates applicant because it would have been obvious to integrate it in the FIFO.

46. Claim 30 is rejected under 35 U.S.C. 103(a) as being unpatentable over Joy in view of Kruse in view of Anderson, as applied above, and further in view of Cutler et al., U.S. Patent No. 5,752,031 (herein referred to as Cutler).

47. Referring to claim 30, Joy in view of Kruse in view of Anderson has taught thread execution control as described in claim 27. Joy in view of Kruse in view of Anderson has not explicitly taught the specifics of claim 30. However, Cutler has taught control logic to:

a) dispatch a packet to a thread. See column 9, lines 63-66, and note that a request packet (and piece of data) is sent to a thread so that space is allocated for it (i.e., it is created and ready to perform work).

b) move the thread from the initialize state to a ready state. See Fig.4.

c) request execution cycles for the thread. Clearly, by being in the ready state, the threads are requesting execution.

d) move the thread to the execute state upon grant by the arbiter of an execution cycle. See Fig.4 and note that an arbiter will choose to either preempt a thread standing by for execution or send a thread to an execute stage 40d.

Art Unit: 2183

e) continue to request execution cycles while the thread is queued in the execute state. Again, the thread will continue to request execution cycles until it completes or some other even happens (preemption or lack of resources).

f) return the thread to the initialize state if there is no latency event, or send the thread to the wait state upon occurrence of a latency event. See Fig.4.

Official Notice is taken that this process is well known and expected in the art. Threads are often find themselves in different states. They can be ready for execution, waiting for resources, executing, etc. This scheme allows for multiple threads to exist at once and for multiple threads to share execution time based on a number of factors (for instance, one thread may be preempted by another thread due to priority). This scheme also allows for the maximization of processor resources because if one thread cannot continue executing (due to stalling), then instead of the CPU sitting idle, another thread may be switched in. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy to include the control logic as taught by Cutler.

48. Claims 1-7, 9-17, and 19-23 are rejected under 35 U.S.C. 103(a) as being unpatentable over Joy, as applied above, in view of Kruse, as applied above.

49. Referring to claim 1, Joy has taught a use of multiple threads including the steps of:

a) providing multiple instruction execution threads as independent processes in a sequential time frame. See Fig.2B and column 6, lines 59-66.

b) Joy has further taught that one of multiple threads may be selected for execution based on priority. See column 4, lines 20-22. Joy has not explicitly taught queuing the multiple execution

Art Unit: 2183

threads. However, Kruse has taught the well-known concept of a priority queue. A priority queue, according to Kruse on pages 369-370, is specifically useful in time-sharing computer systems, which have a number of tasks (threads), each of the tasks having a corresponding priority. The priority queue allows for the finding and removal of the entry having the highest priority. Consequently, since Joy has taught one of multiple tasks are selected based on priority, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy to include a priority queue to hold these tasks because Kruse has taught that the priority queue is a structure which is able to provide such functionality.

c) executing a first thread in the queue. See column 4, lines 20-22. The highest-priority thread that is awaiting processing will be selected for execution.

d) transferring control of the execution to the next thread in the queue upon the occurrence of an event that causes execution of the first thread to stall. See Fig. 2B and column 6, lines 59-66.

e) Joy has not explicitly taught that the multiple threads have overlapping access to the accessible data available in said tree search structure. However, Kruse has taught that search trees are quick and efficient for inserting, deleting, and searching for data. See section 10.2 on pages 444-445. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy such that the threads have access to data available in a tree search structure. Again, one would be motivated to have such a structure because they allow for fast searching, inserting, and deleting of data.

50. Referring to claim 2, Joy in view of Kruse has taught a use of multiple threads as described in claim 1. Joy has further taught that the control of the execution is temporarily transferred to the next thread when execution stalls due to a short latency event, and the control

Art Unit: 2183

is returned to the original thread when the event is completed. See column 2, lines 1-3, column 4, lines 20-22, column 6, lines 59-66, and Fig.2B. Note that on a cache miss, execution control is transferred to the next highest priority thread that is waiting idle. Further note that the cache miss can be considered a short latency event since the applicant has failed to define what constitutes a short latency event. It should be noted that “short” is a relative word, i.e., a “short” event may be a “long” event without a “long” event to compare with. Furthermore, applicant’s “short latency event” is not defined as being 25 cycles or less as applicant argues on page 18 of the appeal brief. Instead, page 4, lines 8-12, of applicant’s specification state “If the latency event is programmed to be short, e.g., 25 machine cycles or less...”. Using an example of 25 machine cycles does not exclude the short latency event from requiring some other amount of cycles larger than 25. Consequently, Joy reads on applicant’s claim. Finally, from Fig.2B, it can be seen that control is returned to the original thread after it is done stalling. The claim does not explicitly state that control is returned to the original thread immediately upon completion of the event.

51. Referring to claim 3, Joy in view of Kruse has taught a use of multiple threads as described in claim 2. Joy has further taught that a processor instruction is encoded to select a short latency event. Recall that Joy has taught thread-switching when a cache-miss occurs. See column 2, lines 2-30. It is inherent that such a short latency event (i.e. a cache miss) would result from trying to load data that is not in the cache, for example. Therefore, load instructions which are encoded in such a way that the processor knows to access cache, would actually select a short latency event upon a cache miss.

Art Unit: 2183

52. Referring to claim 4, Joy in view of Kruse has taught a use of multiple threads as described in claim 1. Joy has further taught that full control of the execution is transferred to the next thread when execution of the first thread stalls due to a long latency event. See column 2, lines 1-3, column 4, lines 20-22, column 6, lines 59-66, and Fig.2B. Note that on a cache miss, execution control is transferred to the next highest priority thread that is waiting idle. Further note that the cache miss can be considered a long latency event since the applicant has failed to define what constitutes a long latency event. Also, the cache miss would result in an access to some slower form of memory (another cache, main memory, disk), which would also be considered long latency events. Furthermore, it should be noted that “long” is a relative word, i.e., a “long” event may be a “short” event without a “short” event to compare with. Furthermore, applicant’s “long latency event” is not defined as being greater than 25 cycles as applicant argues on page 18 of the appeal brief. Instead, page 4, lines 8-12, of applicant’s specification state “On the other hand, if the latency event is programmed to be longer, e.g. over 25 cycles...”. Using an example of being greater than 25 machine cycles does not exclude the long latency event from requiring some other amount of cycles less or equal to 25.

Consequently, Parady reads on applicant’s claim.

53. Referring to claim 5, Joy in view of Kruse has taught a use of multiple threads as described in claim 4. Joy has further taught that a processor instruction is encoded to select a long latency event. Recall that Joy has taught thread-switching when a cache-miss occurs. See column 2, lines 2-30. It is inherent that such a long latency event (i.e. a cache miss and ultimately a slower memory access) would result from trying to load data that is not in the cache,



Art Unit: 2183

for example. Therefore, load instructions which are encoded in such a way that the processor knows to access cache, would actually select a long latency event upon a cache miss.

54. Referring to claim 6, Joy in view of Kruse has taught a use of multiple threads as described in claim 1. Joy has further taught queuing the threads to provide rapid distribution of access to shared memory. See Fig.3, components 330 and 320, and column 9, lines 1-5, and lines 36-40.

55. Referring to claim 7, Joy in view of Kruse has taught a use of multiple threads as described in claim 1. Joy has further taught that the threads have overlapping access to shared remote storage via a pipelined coprocessor by operating within different phases of a pipeline of the coprocessor. See Fig.9 and column 19, lines 6-58. Note that the second processor (coprocessor) is pipelined. Further note that the co-processor threads have overlapping access to main memory via MIU 928 as well as to all caches.

56. Referring to claim 9, Joy in view of Kruse has taught a use of multiple threads as described in claim 1. Joy has further taught that the threads are used with zero overhead to switch execution from one thread to the next. See column 6, lines 34-36, and note that the replication of registers for each thread would result in zero switching overhead, as discussed in column 4, lines 12-16. In addition, Joy makes no mention of switch overhead. Consequently, Joy has taught that the switching requires no overhead.

57. Referring to claim 10, Joy in view of Kruse has taught a use of multiple threads as described in claim 9. Joy has further taught that each thread is given access to general purpose registers and local data storage to enable switching with zero overhead. See column 6, lines 34-36, column 4, lines 12-16, and column 8, lines 59-67. Note that the registers are replicated for

Art Unit: 2183

each thread and each thread is also given its own load and store buffers to hold any pending data coming from or going to memory. Furthermore, from Fig. 7A, it can be seen that the data cache is divided into separate parts for each thread.

58. Referring to claim 11, it has been noted by the examiner that claim 11 claims a processor that uses multiple threads as described in claim 1. Therefore, claim 11 is rejected for the same reasons set forth in the rejection of claim 1.

59. Referring to claim 12, Joy in view of Kruse has taught a processing system as described in claim 11. Furthermore, claim 12 is rejected for the same reasons set forth in the rejection of claim 2.

60. Referring to claim 13, Joy in view of Kruse has taught a processing system as described in claim 12. Furthermore, claim 13 is rejected for the same reasons set forth in the rejection of claim 3.

61. Referring to claim 14, Joy in view of Kruse has taught a processing system as described in claim 11. Furthermore, claim 14 is rejected for the same reasons set forth in the rejection of claim 4.

62. Referring to claim 15, Joy in view of Kruse has taught a processing system as described in claim 14. Furthermore, claim 15 is rejected for the same reasons set forth in the rejection of claim 5.

63. Referring to claim 16, Joy in view of Kruse has taught a processing system as described in claim 11. Furthermore, claim 16 is rejected for the same reasons set forth in the rejection of claim 6.

Art Unit: 2183

64. Referring to claim 17, Joy in view of Kruse has taught a processing system as described in claim 16. Furthermore, claim 17 is rejected for the same reasons set forth in the rejection of claim 7.

65. Referring to claim 19, Joy in view of Kruse has taught a processing system as described in claim 11. Furthermore, claim 19 is rejected for the same reasons set forth in the rejection of claim 9.

66. Referring to claim 20, Joy in view of Kruse has taught a processing system as described in claim 19. Furthermore, claim 20 is rejected for the same reasons set forth in the rejection of claim 10.

67. Referring to claim 21, Joy in view of Kruse has taught a processing system as described in claim 20. Joy has further taught that the local data storage is made available to the processor by providing one address bit under the control of the thread execution control logic and by providing the remaining address bits under the control of the processor. Regarding the data storage (as shown in Fig.8), Joy has taught that the cache is segregated into a first thread's portion and a second thread's portion. Fig.8 also shows the addressing format for accessing the data cache. Note that index field 812 (split into 823 and 824) includes one bit specified by the thread ID (TID), which is an identification tag unique to each thread. See column 3, lines 52-55. This thread ID, when implemented as part of the index field, provides addresses that are only accessible by the thread associated with the corresponding thread ID. The remaining bits are part of a virtual address which is supplied by the processor's load/store units. See column 8, lines 59-67. In addition, recall that there are separate registers for each thread (see column 6, lines 34-36). In Fig.13 and column 27, lines 32-37, Joy has disclosed that each plane (register window)

Art Unit: 2183

1310 represents a separate group of registers within a 3-dimensional register file. There is a one-to-one mapping of register windows to threads, and consequently, thread-switching also results in register window switching so that each thread has its own set of registers. Selecting a window is performed by changing the window pointer. As described in a similar fashion above, it would have been obvious to use the thread ID or a subset of the thread ID to specify which register set will be active at a given time.

68. Referring to claim 22, Joy in view of Kruse has taught a processing system as described in claim 20. Joy has further taught that the processor is capable of simultaneously addressing multiple register arrays, and the thread execution control logic includes a selector to select which array will be delivered to the processor for a given thread. See Fig. 13. Recall from the rejection of claim 21, that each plane 1310 represents an array of registers (register file). From Fig. 13, it can be seen that the register index is decoded such that register N is selected from among each of the arrays. However, the current window pointer 1312 is decoded such that register N from the third window will be selected. Therefore, the current window pointer acts as a selector in that it selects a register window for use by the thread.

69. Referring to claim 23, Joy in view of Kruse has taught a processing system as described in claim 20. Joy has further taught that the local data storage is fully addressable by the processor, an index register is contained within the register array, and the thread execution control has no address control over the local data storage or the register arrays. See Fig. 8 and note the 64-bit indexing format, which, as known in the art, could be stored in one of the 64-bit registers (for indirect addressing purposes) in register file 1300 (Fig. 13). One embodiment, as described in the rejection of claim 21, has part of the index field including a thread ID, which is

Art Unit: 2183

provided based on the currently executing thread. However, according to column 18, lines 21-38, this thread ID tagging can be disabled in cases where native threads and lightweight processes are sharing the same virtual address space while being executed. This allows for a non-segregated cache, which would eliminate the under-utilization of resources resulting from certain threads.

70. Claims 8 and 18 are rejected under 35 U.S.C. 103(a) as being unpatentable over Joy in view of Kruse, as applied above, in view of Parady, as applied above, and further in view of Flynn, as applied above.

71. Referring to claim 8, Joy in view of Kruse has taught a use of multiple threads as described in claim 1.

a) Joy in view of Kruse has not explicitly taught providing a separate instruction buffer for each execution thread. However, Parady has taught the concept of providing separate buffers for each execution thread. See column 5, lines 6-10. Parady has further disclosed that upon a thread switch, the stream of instructions in one of the instruction buffers will simply pick up where it left off. See column 3, lines 51-56. This would eliminate having to flush a single instruction buffer (if only one buffer were used to store instructions for all of the threads) and refilling it with the correct thread's instructions. Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to provide a prefetch buffer for each thread.

b) Joy in view of Kruse in view of Parady has taught providing separate buffers for each execution thread. Joy in view of Kruse in view of Parady has not explicitly taught collecting instructions in a prefetch buffer for its execution thread when the thread is idle and when the

Art Unit: 2183

instruction bandwidth is not being fully utilized. However, Flynn has taught such a concept. See column 4, lines 43-53. Note that when a thread is inactive, a thread's instructions may be fetched so that when the thread is made active, the instruction will be ready for dispatch immediately, thereby increasing efficiency. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy in view of Kruse in view of Parady such that each buffer is a prefetch buffer and instructions for an inactive thread are prefetched into its prefetch buffer.

72. Referring to claim 18, Joy in view of Kruse has taught a processing system as described in claim 11. Furthermore, claim 18 is rejected for the same reasons set forth in the rejection of claim 8.

73. Claims 24 and 25 are rejected under 35 U.S.C. 103(a) as being unpatentable over Joy, as applied above, in view of Parady, as applied above, in view of Flynn as applied above, in view of Kruse, as applied above, and further in view of Anderson, as applied above. In addition, The Free On-Line Dictionary Of Computing (FOLDOC), 1999, is cited as extrinsic evidence showing the meaning of "FIFO" (first-in first-out) and attached to this Office Action.

74. Referring to claim 24, Joy has taught a processor configuration comprising:

- a) a CPU with multiple threads. See Fig.2B and Fig.2C.
- b) an instruction memory. See Fig.3, component 330, and column 9, lines 36-38.
- c) an array of general purpose registers, said array communicating with the CPU. See column 6, lines 34-36.

Art Unit: 2183

d) a local data storage including separate storage space for each thread. See Fig.7A and note that the data cache is divided into separate parts for each thread.

e) a thread execution control for the general register array and the local data storage. As discussed in parts (c) and (d) above, each thread has its own register set as well as its own portion of data cache. Therefore, when threads are switched, it is inherent that some type of thread execution control is used to control access to certain parts of cache and certain register files so that one thread does not interfere with another thread's resources. has not explicitly taught collecting instructions in a prefetch buffer for its execution thread when the thread is idle and when the instruction bandwidth is not being fully utilized. However, Flynn has taught such a concept. See column 4, lines 43-53. Note that when a thread is inactive, a thread's instructions may be fetched so that when the thread is made active, the instruction will be ready for dispatch immediately, thereby increasing efficiency. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy in view of Kruse in view of Parady such that each buffer is a prefetch buffer and instructions for an inactive thread are prefetched into its prefetch buffer.

f) a first coprocessor connecting the CPU to a local data storage. See Fig.9 and note that coprocessor 902 connects the CPU 904 to a local data storage 920.

g) a shared remote storage. See Fig.9 and column 19, lines 6-58. Note that the processor(s) have access to external memory via memory interface unit (MIU) 928.

h) a pipelined coprocessor connecting the shared remote storage and the CPU. See Fig.9 and note that coprocessor 902 connects processor 904 and the shared remote storage via bus 926.

Art Unit: 2183

i) Joy has not explicitly taught a separate queue for each thread between the instruction memory and the CPU. However, Parady has taught the concept of providing separate queues for each execution thread. See column 5, lines 6-10. Parady has further disclosed that upon a thread switch, the stream of instructions in one of the instruction buffers will simply pick up where it left off. See column 3, lines 51-56. This would eliminate having to flush a single instruction buffer (if only one buffer were used to store instructions for all of the threads) and refilling it with the correct thread's instructions. Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to provide a queue for each thread. In turn, Joy in view of Parady has not explicitly taught collecting instructions in a prefetch buffer for its execution thread when the thread is idle and when the instruction bandwidth is not being fully utilized. However, Flynn has taught such a concept. See column 4, lines 43-53. Note that when a thread is inactive, a thread's instructions may be fetched so that when the thread is made active, the instruction will be ready for dispatch immediately, thereby increasing efficiency. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy in view of Parady such that each queue is a prefetch queue and instructions for an inactive thread are prefetched into its prefetch queue.

j) Joy has further taught that one of multiple threads may be selected for execution based on priority. See column 4, lines 20-22. Joy has not explicitly taught that said thread execution control includes a priority FIFO buffer to store thread numbers. However, Kruse has taught the well-known concept of a priority queue (also known as a priority FIFO since "FIFO" and "queue" are interchangeable according to FOLDOC). A priority FIFO, according to Kruse on pages 369-370, is specifically useful in time-sharing computer systems, which have a number of



Art Unit: 2183

tasks (threads), each of the tasks having a corresponding priority. The priority FIFO allows for the finding and removal of the entry having the highest priority. Consequently, since Joy has taught one of multiple tasks are selected based on priority, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy to include a priority FIFO to hold these tasks because Kruse has taught that the priority FIFO is a structure which is able to provide such functionality. In addition, although Parady in view of Kruse has not taught actually storing thread numbers in the priority FIFO, Anderson has taught such a concept. See column 6, lines 42-45. A person of ordinary skill in the art would've recognized that there must be an efficient way to distinguish the tasks from each other. Anderson has taught that thread Ids (numbers) are stored in the priority FIFO. This is applicable to Joy because Joy employs thread Ids (TIDs). See Fig.5, lines 407, for instance. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy in view of Kruse such that thread numbers are stored in the priority FIFO.

75. Referring to claim 25, Joy in view of Parady in view of Flynn in view of Kruse and further in view of Anderson has taught a processor configuration as described in claim 24. Joy has further taught that the thread execution control includes:

- a) a plurality of thread control state machines, one for each execution thread. See Fig.3, components 310 and 312.
- b) an arbiter, responsive to signals from the FIFO buffer and the state machine to determine thread execution priority. It is inherent that if one of a plurality of threads is selected, then some component, more specifically, an arbiter, must determine which thread to select. The highest-priority thread that is awaiting processing will be selected for execution. See column 4, lines 17-

Art Unit: 2183

27. This selection is based on signals from the state machines and signals from the FIFO. More specifically, the state machines provide instructions (i.e., instruction signals) to the pipeline (Fig.3 and column 9, lines 22-25) which will cause the pipeline to stall or not stall (for instance, if the instruction is a load, it will cause the processor to either miss or hit the cache). Once a switch is necessary based at least in part on the signals from the state machines, the priority value signals stored in the FIFO must be available so that the highest priority thread is selected.

### *Conclusion*

Any inquiry concerning this communication or earlier communications from the examiner should be directed to David J. Huisman whose telephone number is (703) 305-7811. The examiner can normally be reached on Monday-Friday (8:00-4:30).

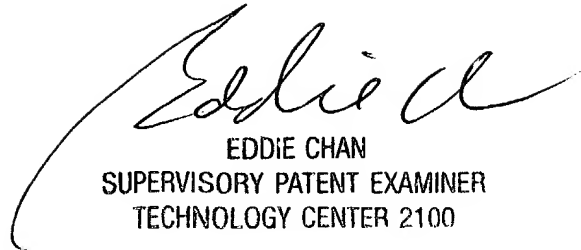
If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Eddie Chan can be reached on (703) 305-9712. The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

DJH

Art Unit: 2183

David J. Huisman  
September 14, 2004



EDDIE CHAN  
SUPERVISORY PATENT EXAMINER  
TECHNOLOGY CENTER 2100